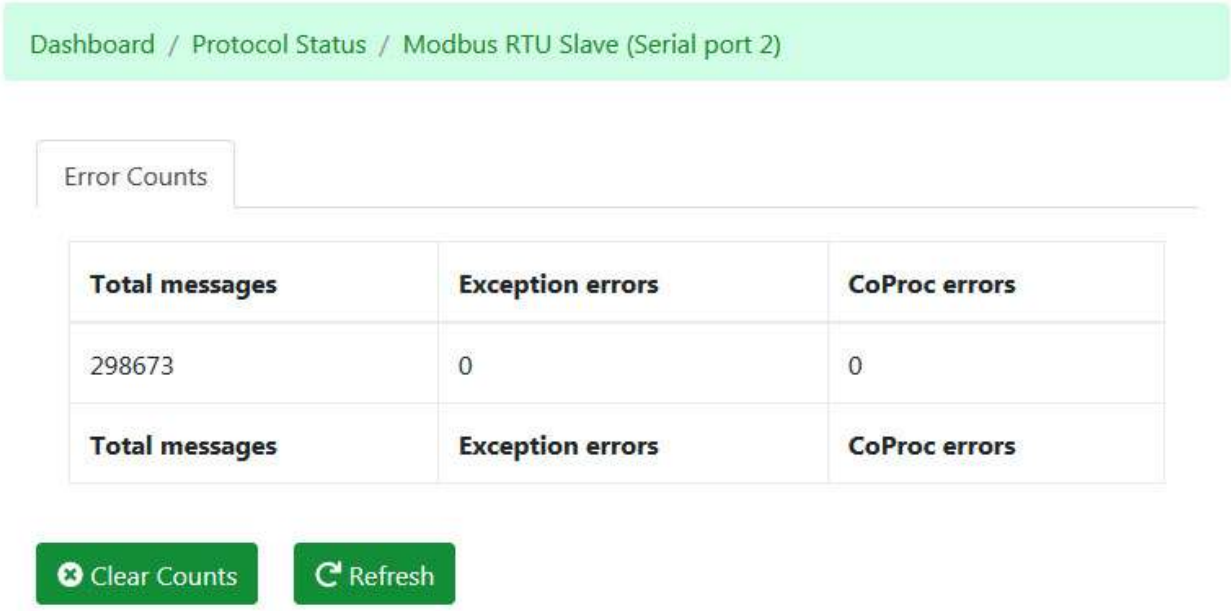
BB4-8422 Modbus RTU Slave

Modbus RTU Slave Error Counts



- Click the Clear Counts to reset the counts to zero.
- Click the Refresh button to update the counts.

The error counts illustrated on this page refer to errors that occurred when remote clients or masters attempted to read/write registers in this IoTServer device.

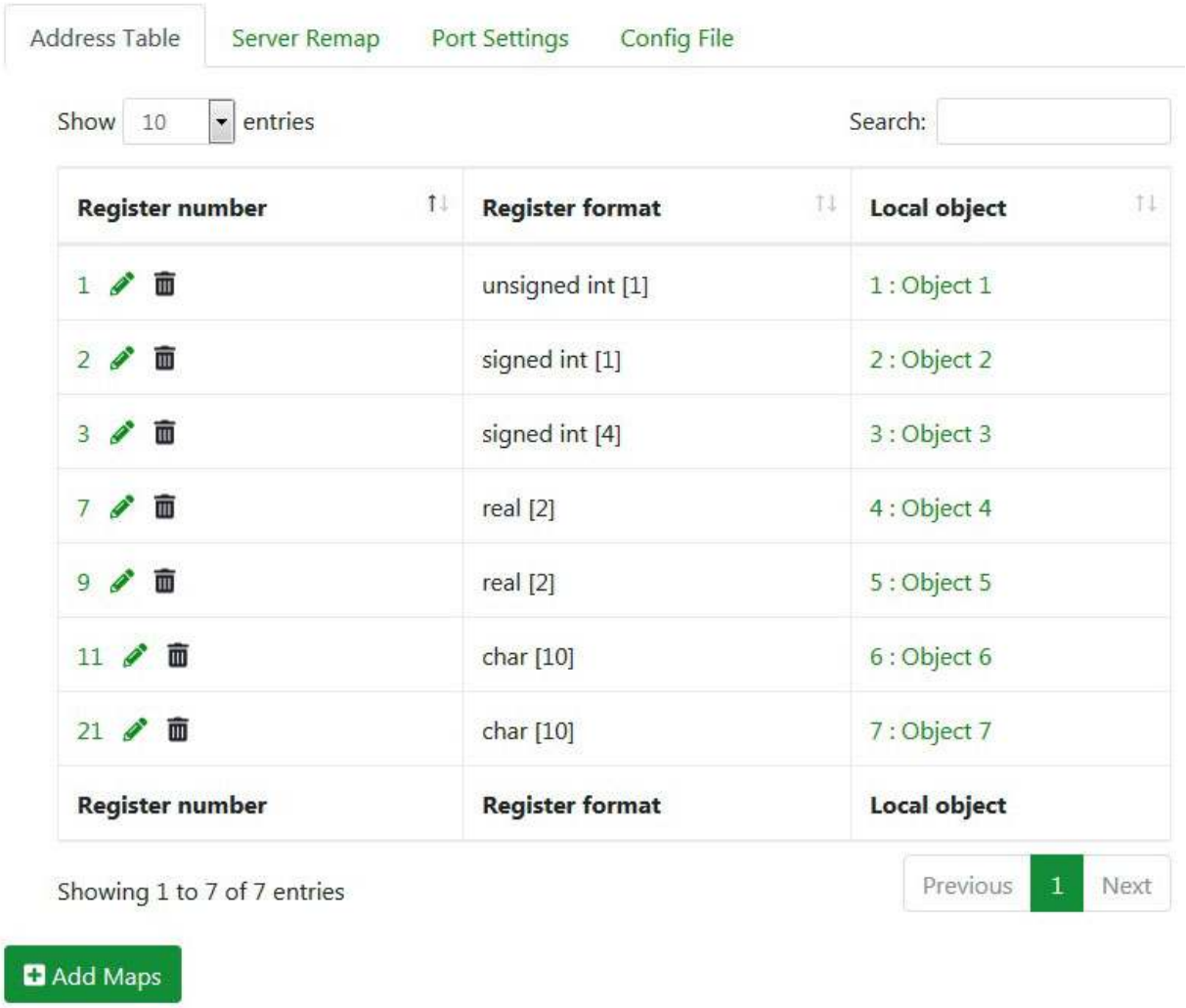
- Total message count is the total number of messages that have been received by this server.
- Exception errors are errors in the request which have been reported back to the client/master.
- CoProc errors are internal errors in communication between the two processors.

Modbus RTU Slave Address Table

The server (slave) address table is used to define the set of Modbus registers that will be visible to other Modbus clients (masters) when the IoTServer is acting as a Modbus server (slave). The set of Modbus registers that exist in the IoTServer can be defined only once, but you have the option of remapping them to different register numbers that will be accessed by secondary ports on the device. Most often you would use the same register numbers whether they are accessed from Modbus TCP or Modbus RTU, but it is possible to use a completely different set of numbers for each different network port.

The address table can only be defined by the Primary Server. All other instances of Modbus servers or slaves are secondary, and will access the same set of Modbus registers with the option of remapping them to different numbers.

The address table as it will appear for the Primary Server is illustrated below.



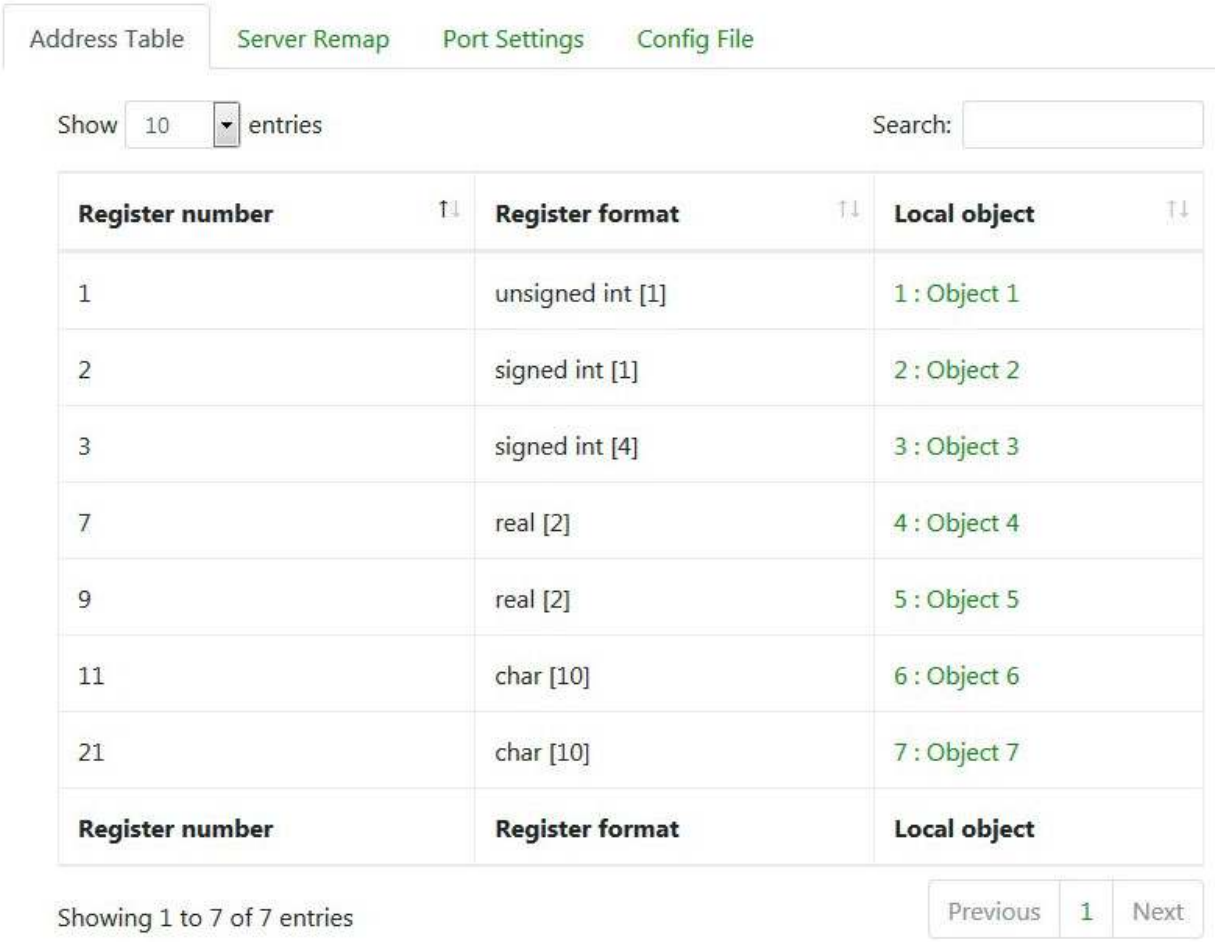
- Click on the register number or pencil icon to jump to the map editing page for this register.
- Click on the trash can icon to delete this Modbus register.
- Click the Add Maps button for a special version of the map editing page.

The address table maps local objects to Modbus registers. Note that in the above example, the Modbus registers skip one or more numbers in some instances. Modbus holding registers are by definition strictly a 16-bit entity. When a data element occupies more than 16 bits, it requires multiple Modbus registers. Therefore a 32-bit value occupies 2 registers, while a 20-character

string will occupy 10 registers (character strings are packed 2 characters per register). The number of Modbus registers assigned to each entry in the table is displayed in brackets in the format column.

Modbus addresses generally refer to holding registers. However, the server will provide access to the same register numbers as input registers (for reading only), or as coils or discrete inputs (for reading only). If a single bit register type such as coil is used to access a register number, the value will be zero if the local object contains a zero, and will provide a one if the local object contains anything other than zero.

The address table as it will appear for all Secondary Servers is illustrated below. Note that the editing icons and buttons are not available in the secondary address table.

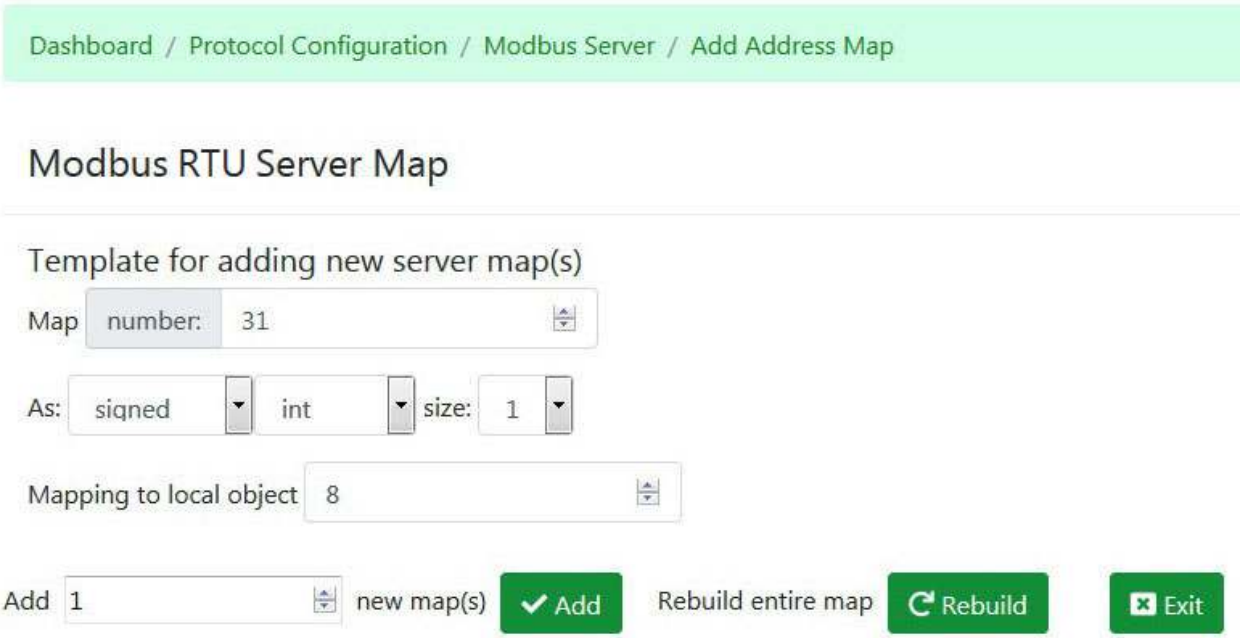


Modbus RTU Slave Address Table Edit

There are two forms of the Modbus server address edit page. When you click on the pencil icon on the address table page to change a map, you will arrive at the following version of the address table edit page:

- Click Save to save settings shown on this page, or Exit to return to the table without change.

If you click on the Add Maps button at the bottom of the address table page, you will arrive at this version of the address table edit page:



- Click Add to add one or more maps resulting in the addition of one more more Modbus registers.
- Click Rebuild to rebuild the entire address table (see additional comments below).
- Click Exit to return to address table without any changes.

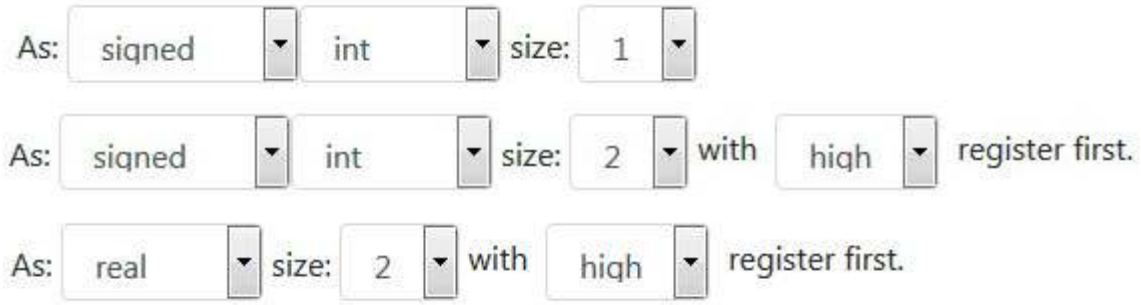
The parameters for either editing or adding Modbus registers to the address table are as follows.

The following line will indicate "number", "address", or "Modicon register", depending on your preference which you set under User Settings.

Register Number or Address – Enter the number (starting at 1) or raw address (starting at 0) as applicable. Do NOT enter 40001 for holding register 1 if you have not selected Modicon as the display format in your User Settings.

Modicon Register – Enter numbers like 40001 for the first holding register if you have selected Modicon representation in your User Settings.

The options available on the next line will vary depending on selections made. The following are a few examples.



Register Format – Select the format of the data contained in the Modbus register(s), not used by the protocol, but used by the gateway to interpret what the raw increments of 1 or 16 bits should mean. Select format from the following table.

Format Label	Format description
"None"	No format defined
"Bit"	Single bit, used ONLY for Register Type Coil or Disc
"Int"	Integer (size and whether signed are defined by labels below)
"Real"	Floating point (single or double precision)
"Char"	Character string with 2 ASCII characters per register
"Mod10"	Mod10 format, can be 2, 3, or 4-register, specific to Schneider Electric meters

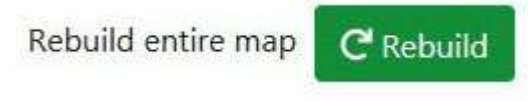
Register Size – Register size refers to the number of consecutive input or holding registers that should be read for a value greater than 16 bits. A 16-bit value would have size of 1, a 32-bit value would have size of 2, and a 64-bit value would have size of 4. Single precision Real (32-bit IEEE 754 floating point) would be size 2, and double precision Real (64-bit IEEE 754 floating point) would be size 4. If format is Mod10, then valid sizes are 2, 3, or 4 – check manufacturer's documentation if Mod10 is noted. Register "size" for a character string will be character count divided by 2 (plus 1 of string length is an odd number). Register Size is not used for Coil or Disc types.

IMPORTANT: If the register size is more than 1, then the next assigned register address must be incremented by this count.

Unsigned – Select signed or unsigned. Defaults to signed integer. Has no effect on Register Format other than Int.

Endian Selection – Used when Register Size is greater than 1 to indicate what order the registers should be interpreted in. Select "low" to indicate that the lowest numbered register contains the least significant portion of data. Select "high" to indicate that the lowest numbered register contains the most significant portion of data. Although Modbus protocol itself is not inherently "Little Endian", many devices operate that way due to Intel processors being inherently Little Endian. Modbus protocol does not stipulate what the register order should be when multiple registers are treated as a single data entity. Therefore, the user is required to pay attention to this.

Local Object – Specifies the local object number that contains the data that will be provided to external Modbus clients or masters requesting this Modbus "register".

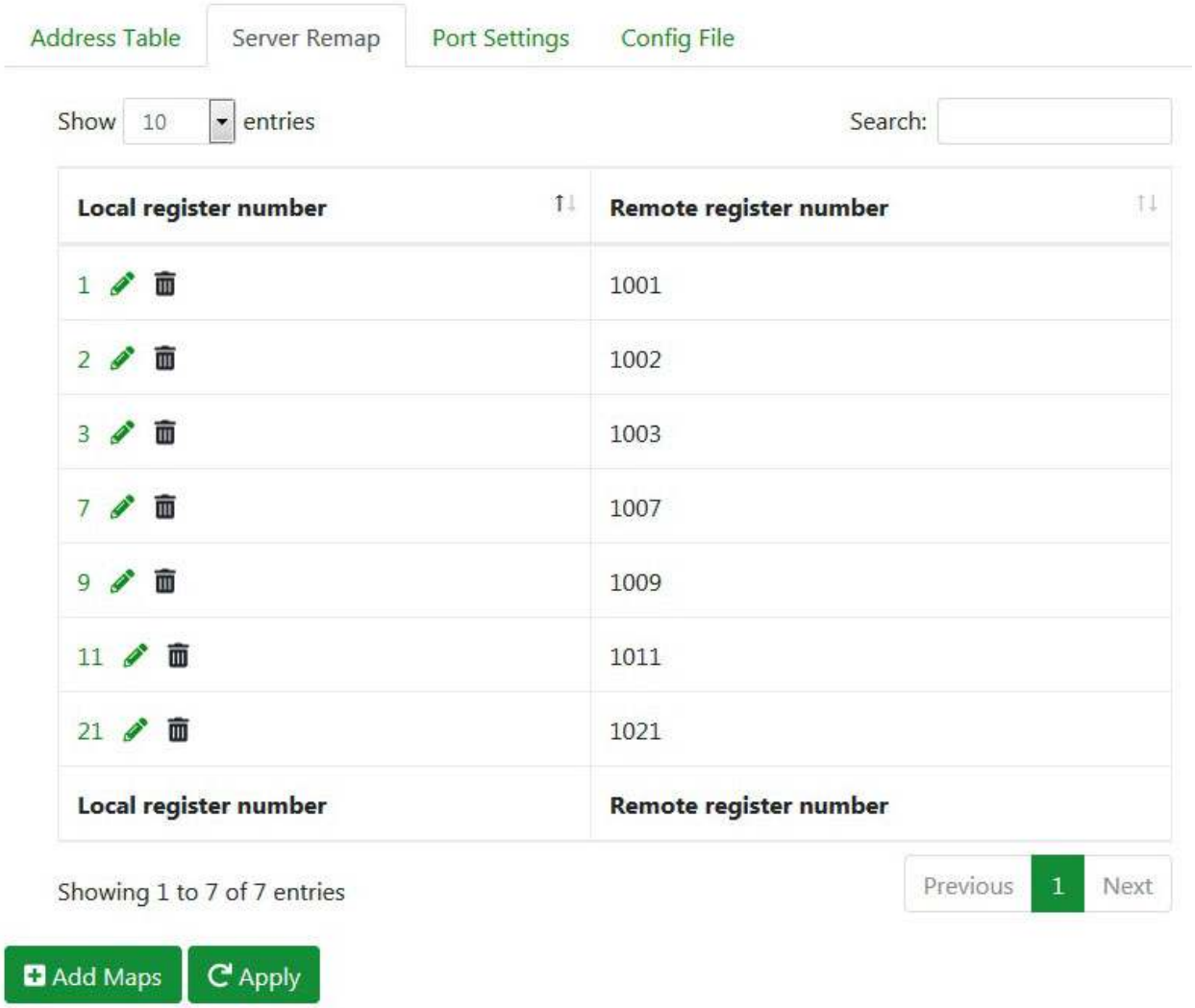


The Rebuild button is a special case button for rebuilding the entire address map. This is the easiest way to create a default address table. Only the format information is used in this case, for example:

When using the Rebuild button, the starting number will always be Modbus register 1, and the starting local object will always be object 1. The rebuild will automatically assign the correct number of Modbus registers, as determined by the format, to each local object. Modbus register numbers will be assigned to all available local objects.

Modbus RTU Server Remap Table

This page is where you have the option of remapping the Modbus addresses or register numbers that some other Modbus client or master would see when polling this device. Start by reviewing the Slave Address Table page if you haven't already.



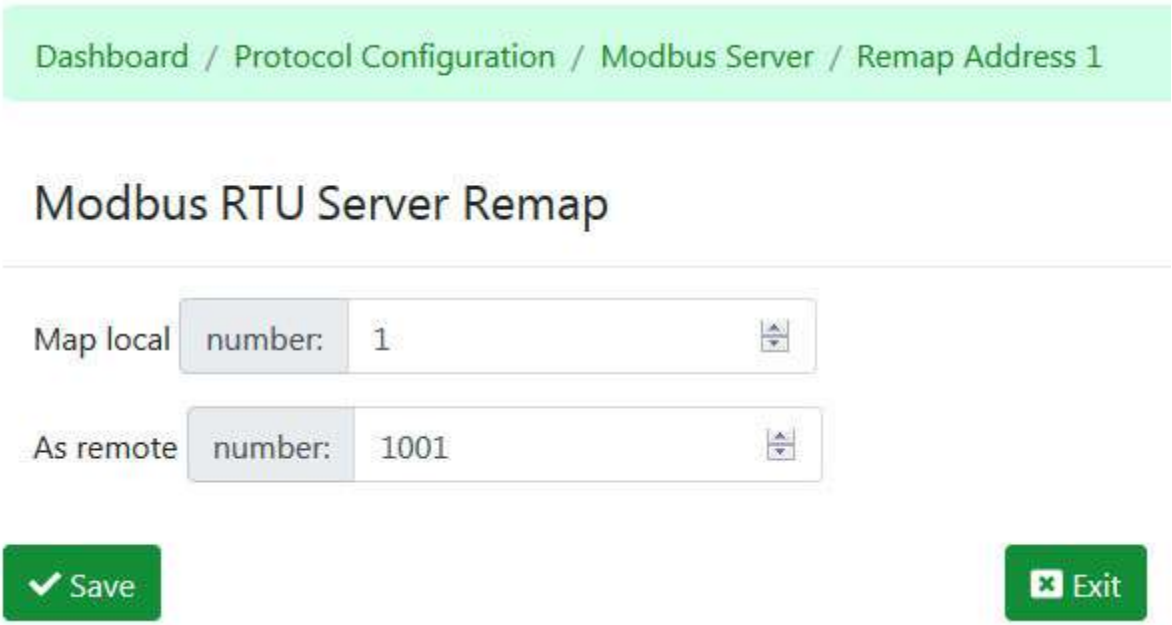
- Click register number or pencil icon to edit the remap entry.
- Click the trash can icon to delete this remap entry.
- Click the Add Maps button for the "add" version of the edit page.
- Click the Apply button to reload the server's remap table as displayed.

Server Remapping Not Enabled

Note: If the server remapping option is not enabled for this server in its Task Configuration, then the remap table will be empty and the Add Maps and Apply buttons will be replaced with a message saying "Server Remapping Not Enabled".

Modbus RTU Server Remap Table Edit

The edit page for server address map is very simple.



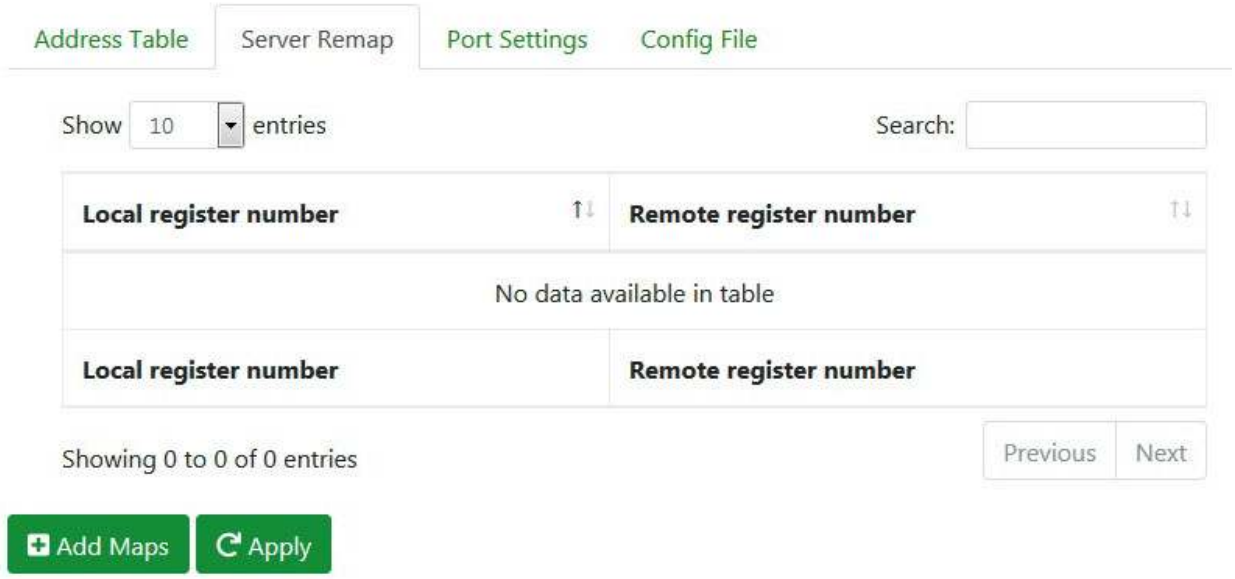
Map local will indicate "number", "address", or "Modicon register", depending on your preference which you set under User Settings.

Register Number or Address – Enter the number (starting at 1) or raw address (starting at 0) as applicable. Do NOT enter 40001 for holding register 1 if you have not selected Modicon as the display format in your User Settings.

Modicon Register – Enter numbers like 40001 for the first holding register if you have selected Modicon representation in your User Settings.

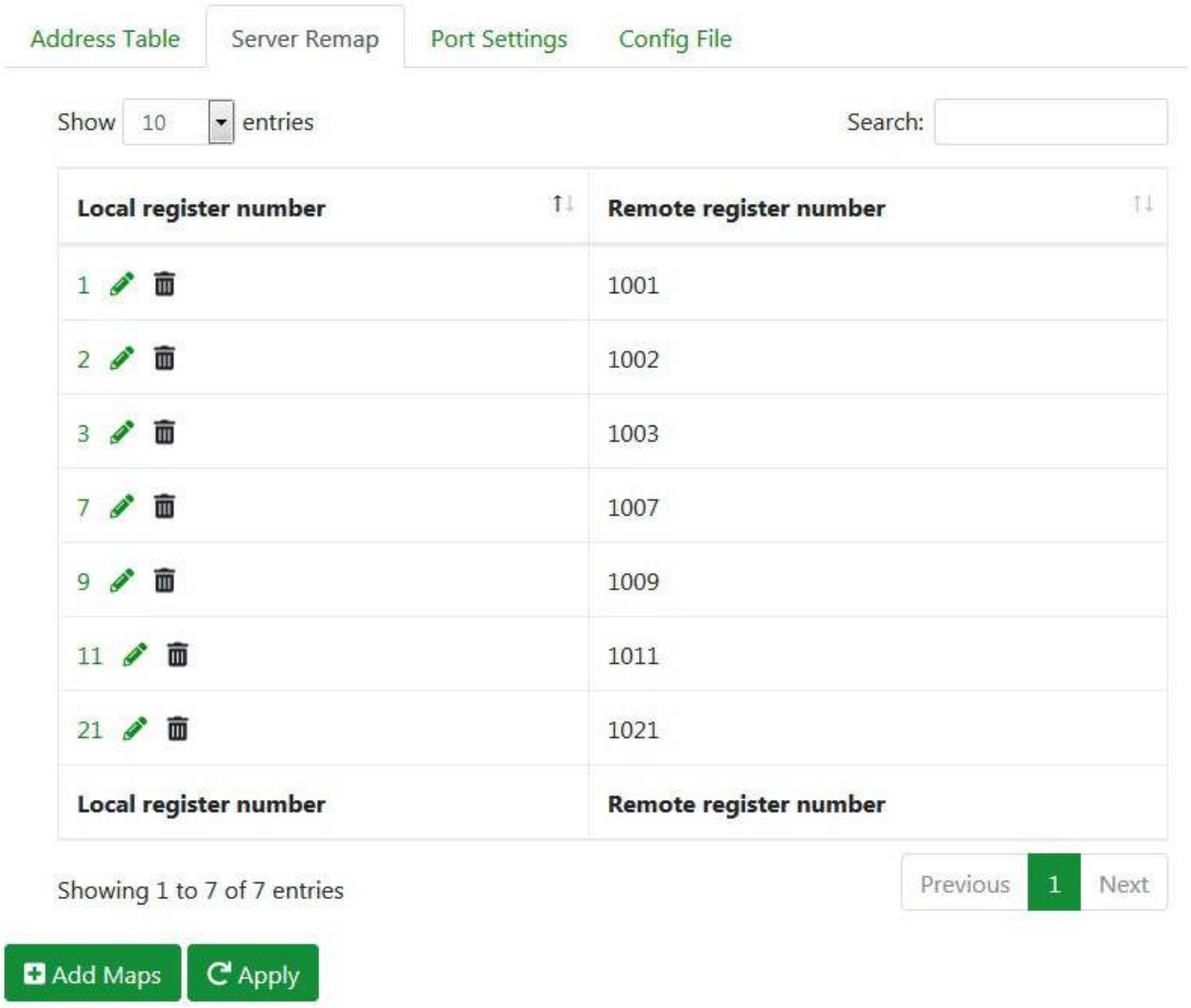
As remote – The remote Modbus register address or number that this local address should be viewed as by remote clients or masters. Follow the same rules for number, address, or Modicon register as for the local value.

The Add Maps version of the remap edit page works as follows. Let's assume you start with the

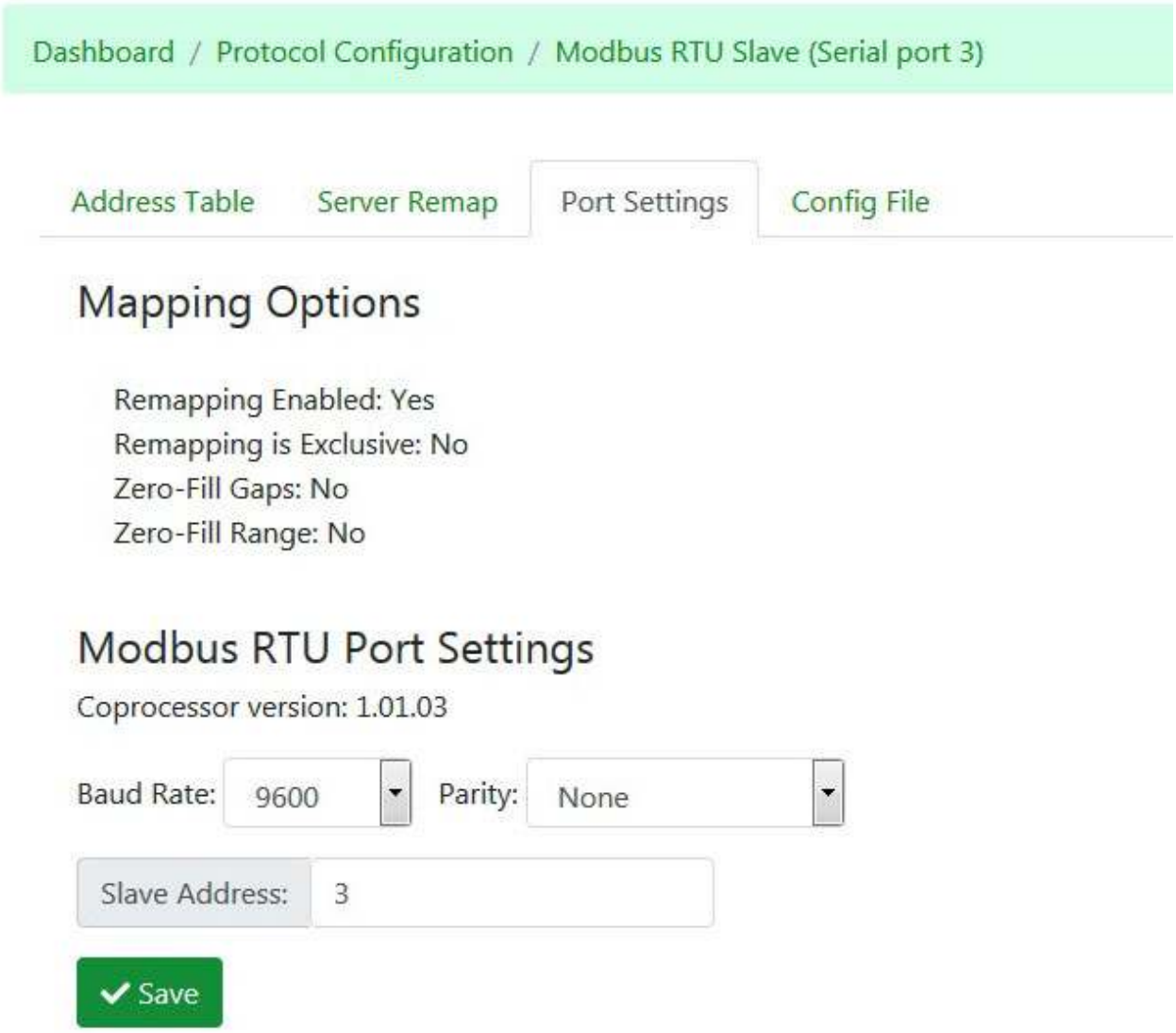


The add/edit page is very simple. Follow the same guidelines for local register numbers as noted above for editing. On this version of the edit, you provide an offset instead of specific register number. Each remapped number will be the local number plus offset. The other difference on the Add page is that you also have an ending local number. Click Add to add new entries for the entire range of addresses.

The result in our example is:



Modbus RTU Slave Port Settings

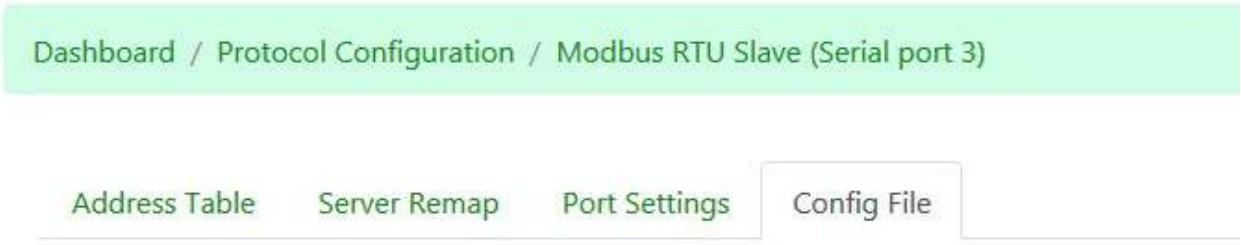


Select the baud rate and parity for the Modbus RTU slave port. These settings are made individually for each Modbus RTU port.

Provide a slave address for the Babel Buster 4. This is the address that other Modbus masters will use when communicating with this Babel Buster 4 operating as a Modbus slave.

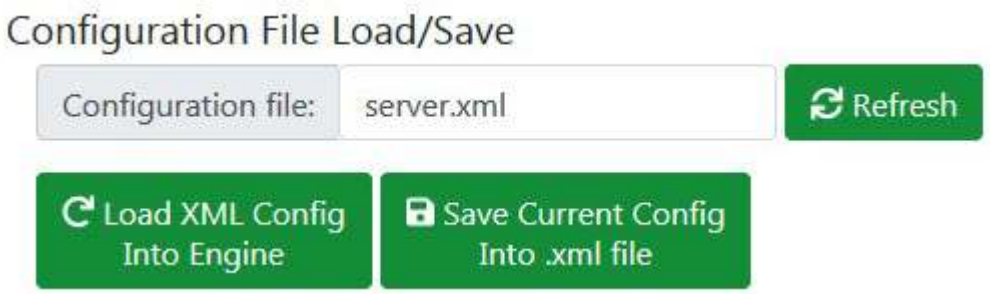
When the RTU port is slave, additional information about its remapping parameters will be displayed. These can only be changed via the *Task Configuration*.

Modbus RTU Slave Config File



All of your configuration information is stored in an internal database each time you click the Save button on any page where configuration entries may be made. To make configuration portable from one device to another, and for purposes of retaining a backup copy, the configuration information may be exported and imported as XML or CSV files. This page is where your configuration file management takes place.

It is important to note that the XML file saved within any one client/server function will contain the configuration information for only that function. Depending on overall system configuration, a complete backup may involve more than one XML or CSV file.



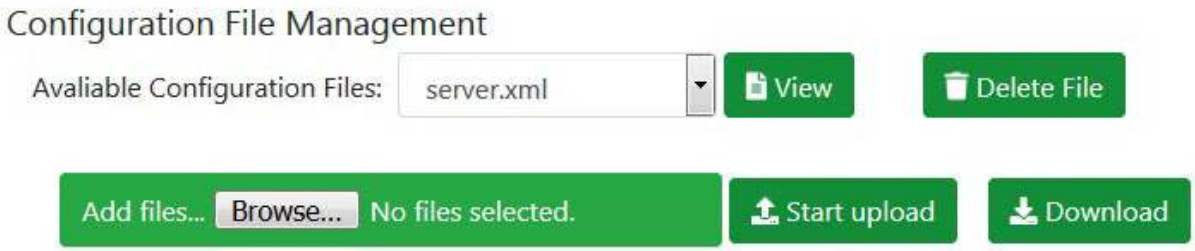
XML Files: When an XML file has been selected, click the Load button to clear the configuration database and reload configuration from the given XML file.

Select an existing name to overwrite or enter a new file name, and then click Save to write the current configuration to the file in XML format.

You may type in a new name in the file name window for purposes of saving a new file. If you click the Refresh button, the file name will be restored to the name currently loaded into the client. The name could have been changed by selecting a file from the list below, or by typing in a new name. If the displayed name has not yet been used, then Refresh will restore the file name to what was most recently loaded.

CSV Files: If a CSV file is selected, the Load and Save buttons will change into load/save CSV buttons. When a CSV file has been selected, click the Load button to clear the configuration database and reload configuration from the given CSV file.

Select an existing name to overwrite or enter a new file name, and then click Save to write the current configuration to the file in CSV format.



The drop-down list will show a list of all configuration files currently found in the device's configuration folder. When you select an XML or CSV file from this list, the name will be copied to the Load/Save section of this page for pending load or save.

You may view the selected file by simply clicking View. You can delete the file by clicking Delete.

You may upload files to the IoTServer from your PC. Start by clicking Browse, and then use the browser's file dialog to locate the file on your PC. Once a file is selected on your PC, click the "Start upload" button to initiate the transfer.

You may also download files from the IoTServer to your PC. Click the Download button to transfer the selected file to your PC.

Any time an XML or CSV file is loaded, an error log file is generated. The error log file will be given the same name as the loaded file, but with ".err" as the suffix instead of ".xml" or ".csv". You may view the error log by selecting it from the list and clicking View.

Status is normally displayed in a message box at the top of the screen when the load or save operation is complete. But if you want to double check the status of the previous file operation, click Check Status.

The task (client or server) needs to be suspended while a file load operation is in progress to prevent acting on any partial configurations. This suspend/resume operation will normally happen automatically as part of the sequence invoked by the Load button when loading an XML file. The task must be explicitly suspended here for importing a CSV file. The Suspend button will become a Resume button when the task is suspended. Click Resume to continue operation. The current status is always displayed here.

Modbus RTU Slave XML Files

Example XML File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
<server_regs>
<reg addr="0" format="bit" size="1" lowfirst="1" objnum="1"/>
<reg addr="1" format="int" size="1" lowfirst="1" objnum="2"/>
<reg addr="2" format="int" size="2" lowfirst="1" objnum="3"/>
<reg addr="4" format="int" size="4" lowfirst="1" objnum="4"/>
<reg addr="8" format="int" size="1" unsigned="1" lowfirst="1" objnum="5"/>
<reg addr="9" format="int" size="2" unsigned="1" lowfirst="1" objnum="6"/>
<reg addr="11" format="int" size="4" unsigned="1" lowfirst="1" objnum="7"/>
<reg addr="15" format="real" size="2" lowfirst="1" objnum="8"/>
<reg addr="17" format="real" size="4" lowfirst="1" objnum="9"/>
</server_regs>
<remap_regs>
<map localAddr="0" remoteAddr="100"/>
<map localAddr="1" remoteAddr="101"/>
<map localAddr="2" remoteAddr="102"/>
<map localAddr="4" remoteAddr="104"/>
<map localAddr="8" remoteAddr="108"/>
<map localAddr="9" remoteAddr="109"/>
<map localAddr="11" remoteAddr="111"/>
<map localAddr="15" remoteAddr="115"/>
```

<map localAddr="17" remoteAddr="117"/>
</remap_regs>
</configuration>

Modbus RTU Slave <server_regs> Section

Each line in the server_regs portion of the XML file defines one locally known Modbus "register" which may actually span multiple Modbus addresses.

NOTE: Register configuration does not specify a Modbus register type. This is because all register addresses in the IoT Server can be accessed as any of the standard register types (coil, discrete input, input register, or holding register). Therefore, the value at input register address 0 and holding register address 0 will be the same value. When referenced as a coil or discrete input the same address will appear as 0 if the value is 0, or 1 if the value in anything non-zero. All local registers are 16 bits regardless of how accessed. The 16-bit value will be processed as 0 or 1 for coil or discrete input. Therefore, writing a 1 to an address as a coil will result in a 16-bit integer 1 when the same address is accessed as a holding register.

IMPORTANT: There can be only one definition of a primary register map, but there can be multiple instances of a Modbus server. The secondary servers can only remap the registers defined by the primary server. The primary definition is found in the <server_regs> section while remapping is found in the <remap_regs> section. A secondary server will disregard the <server_regs> section. The primary server can be either a Modbus RTU Slave or Modbus TCP Server.

XML attributes for each register:

addr="n" – (Json "address", integer) – Modbus address in the range of 0..65535. Note that Modbus address 0 is most often referred to as "Register 1" or Modicon 40001 if it is a holding register.

format="xxx" (Json "format", character string, and "formatCode", integer) – Specifies the Modbus format in which data should be interpreted. Valid formats for XML and Json are shown below:

XML value	Format description
"none"	No format defined
"bit"	Single bit (coil, discrete only)
"int"	Integer (16-, 32-, or 64-bit)
"real"	Floating point (single or double)
"char"	ASCII character string

“mod10” Schneider Electric Mod10 format

size="n" – (Json “size”, integer) – Specifies the number of 16-bit address locations that make up this “register”. Valid sizes by register format are as follows:

Type	Number of registers
Bit	1
Integer	1, 2, 4 (for 16-, 32-, 64-bit)
Real	2, 4 (for single, double precision)
Character	1..63 (registers - 2 characters per register)
Mod10	2, 3, 4

Note: The maximum size for ‘char’ is 63 registers, which means 126 characters is the maximum string length. Size is in registers, not characters.

unsigned=”n” – (Json “unsigned”, integer) – Applies only to integer, and specifies that the integer should be treated as unsigned in “n” is “1”. Integers default to signed (n=0).

lowfirst=”n” – (Json “littleEndian”, integer) – Applies to any register greater than 16 bits (except “char”), meaning the data value consists of 2 or more 16-bit registers. When lowfirst n=1, the least significant data will be found in the lowest numbered or first Modbus register address of the “register” that spans multiple addresses. When lowfirst n=0 (or omitted), the most significant data will be found in the lowest or first Modbus register address. For “char” registers, the start of the string will always be in the lowest or first Modbus address.

objnum=”n” – (Json “localObject”, integer) – Specifies the IoT Server global data object that will be provided via this Modbus register. The global object data type and Modbus register data format do not need to match. Data will be converted as applicable, including numeric to character string conversion.

Modbus RTU Slave <remap_regs> Section

Each line contains simply a local address and a remote address. The local address is expected to reference a register previously created in the server_regs part of the file (by the primary server). The remote address is any valid Modbus address that will be used to reference this local register. There may fewer remap_regs entries than server_regs entries, but generally not more.

localAddr=”n” – (Json “localAddr”, integer) – Specifies local Modbus address, namely the Modbus address established by the Register Configuration above.

remoteAddr="n" – (Json "remoteAddr", integer) – Specifies the address of the same register as seen by a remote client (master) querying our local Modbus device.

Modbus RTU Slave CSV Files

The format of a CSV file is the same for both Modbus RTU Slave and Modbus TCP Server. The term "Server" is synonymous with "Slave".

Modbus Server Example – Primary Server

The following illustrates a CSV file for the primary instance of Modbus Server, which can be assigned to either an RTU or TCP channel. There can be only one instance of a primary server, and this instance establishes the correlation of local objects to Modbus registers. Secondary instances of Modbus servers will refer to the primary server's mapping of objects to register addresses, optionally with remapping (see below).

```
Begin,Modbus,ServerMaps
LocalObj,RegAddr,RegFormat,RegSize,Unsigned,LittleEnd
1,0,INT,1,Y,N
2,1,INT,1,N,N
3,2,INT,4,N,Y
4,6,REAL,2,N,Y
5,8,REAL,2,N,Y
6,10,CHAR,10,N,N
7,20,CHAR,10,N,N
End
```

MODBUS (server/slave) SERVERMAPS Section

LocalObj – Specifies the local object number that contains the data that will be provided to external Modbus clients or masters requesting this Modbus "register".

RegAddr – Raw 0-indexed address to be assigned to this register.

RegFormat – Format of the data contained in the Modbus register(s), not used by the protocol, but used by the gateway to interpret what the raw increments of 1 or 16 bits should mean. Select format from the following table.

Format Label	Format description
"None"	No format defined
"Bit"	Single bit, used ONLY for RegType Coil or Disc

"Int"	Integer (size and whether signed are defined by labels below)
"Real"	Floating point (single or double precision)
"Char"	Character string with 2 ASCII characters per register
"Mod10"	Mod10 format, can be 2, 3, or 4-register, specific to Schneider Electric meters

RegSize – Register size refers to the number of consecutive input or holding registers should be read for a value greater than 16 bits. A 16-bit value would have size of 1, a 32-bit value would have size of 2, and a 64-bit value would have size of 4. Single precision Real (32-bit IEEE 754 floating point) would be size 2, and double precision Real (64-bit IEEE 754 floating point) would be size 4. If format is Mod10, then valid sizes are 2, 3, or 4 – check manufacturer's documentation if Mod10 is noted. Register "size" for a character string will be character count divided by 2 (plus 1 of string length is an odd number). RegSize is not used for Coil or Disc types.

IMPORTANT: If the register size is more than 1, then the next assigned register address must be incremented by this count.

Unsigned – Indicate "Y" if unsigned, or "N" if signed. Defaults to signed integer. Has no effect on RegFormat other than Int.

LittleEnd – Used when RegSize is greater than 1 to indicate what order the registers should be interpreted in. Enter "Y" to indicate that the lowest numbered register contains the least significant portion of data. Enter "N" or omit to indicate that the lowest numbered register contains the most significant portion of data. Although Modbus protocol itself is not inherently "Little Endian", many devices operate that way due to Intel processors being inherently Little Endian. Modbus protocol does not stipulate what the register order should be when multiple registers are treated as a single data entity. Therefore, the user is required to pay attention to this.

Modbus Server Example – Secondary Server

The following illustrates the CSV file or section for remapping register addresses in an instance of a Modbus server. The remapping can be optionally included in the primary instance, but is usually more commonly used in a secondary instance of server. The remapping effectively creates a different view of the same set of registers defined by the primary server. For example, in the CSV below, Modbus register 0 is seen at address 100 by any client requesting registers from this instance of the server. Different instances of the server will normally run on different ports in the system.

```
Begin,Modbus,ServerRemaps
LocalAddr,RemoteAddr
0,100
1,101
```

2,102
6,106
8,108
10,110
20,120
End

MODBUS (server/slave) SERVERREMAPS Section

LocalAddr – The local Modbus register addressed assigned by the primary server. This address must exist in the register assignments made by the primary server.

RemoteAddr – The remote Modbus register address that this local address should be viewed as.